CI1057: Algoritmos e Estruturas de Dados III
Árvores Binárias Balanceadas - Árvores AVL - Remoção

Profa. Carmem Hara

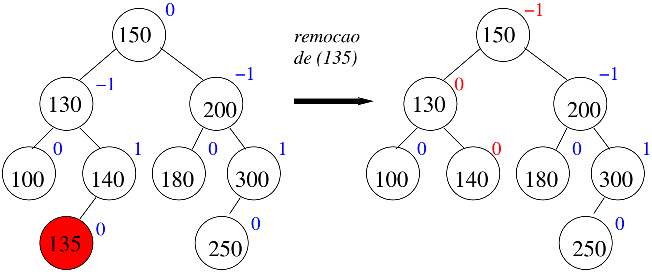
Departamento de Informática/UFPR

21 de março de 2024

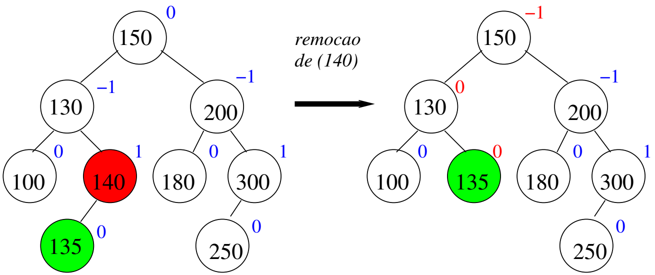
Relembrando as Operações de Inserção em uma árvore AVL

- inserção dos valores: 100 200 300 150 130 140

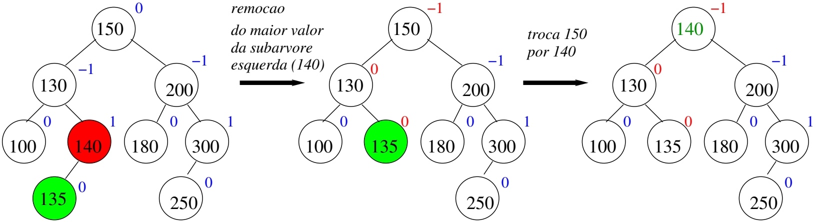
Remoção de nodo folha



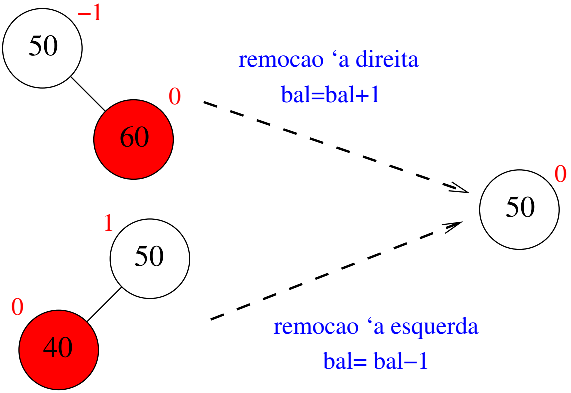
Remoção de nodo com um único filho



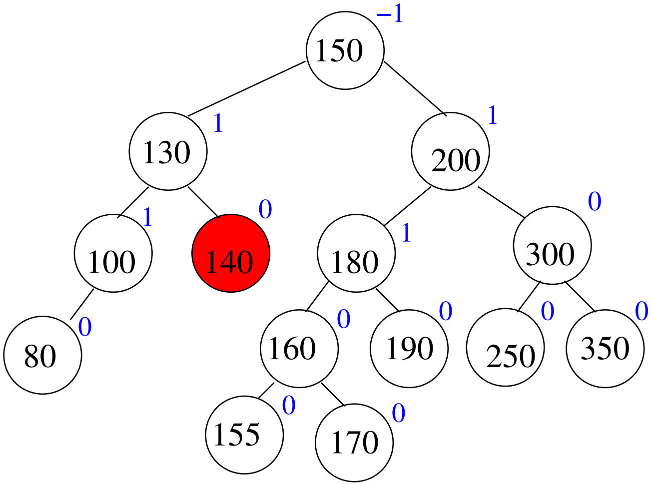
Remoção de nodo com 2 filhos não nulos



Remoção: alteração do balanceamento



Remoção com rebalanceamento



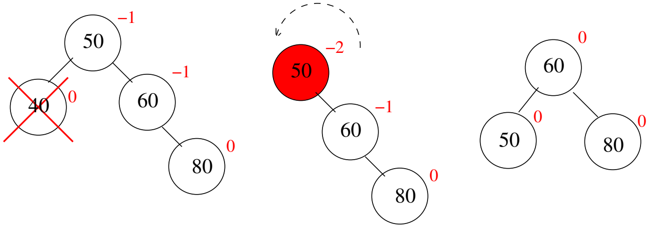
Remoção com rebalanceamento

- um único balanceamento não necessariamente balanceia a árvore toda
- diferente da inserção!!
- quando a altura da árvore muda é possível que os ancestrais do nodo balanceado também fiquem desbalanceados

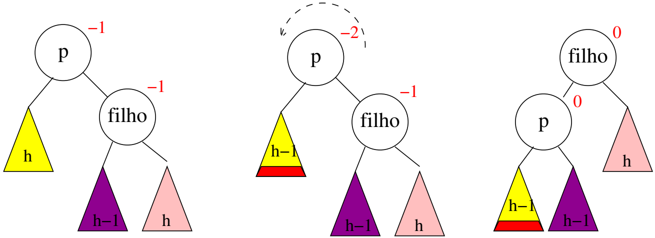
Remoção da AVL

1. nodoK = busca(k, arvore)
2. se nodoK tem 2 filhos
   então nodoRem= nodo com maior valor na subárvore esquerda de nodoK
   senão nodoRem= nodoK
3. remover nodoRem da árvore,
   alterando o balanceamento dos seus ancestrais (a):
   se chave(nodoRem) $<$ chave(a)
   bal(a) = bal(a) - 1
   senão
   bal(a) = bal(a) + 1
   se o ancestral a ficar desbalanceado
   então balancear a
4. se o balanceamento de a tornar-se diferente de zero
   então não alterar mais os balanceamentos dos seus ancestrais
5. se o nodoRem for diferente de nodoK
   então alterar o valor de nodoK pelo valor armazenado em nodoRem

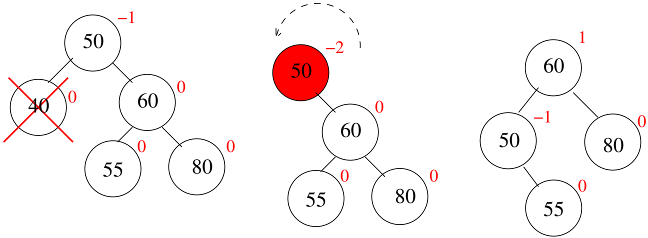
Remoção - Caso Dir-Dir - bal(filho)=-1



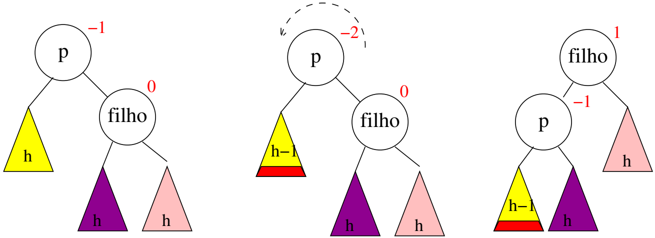
De forma generica:



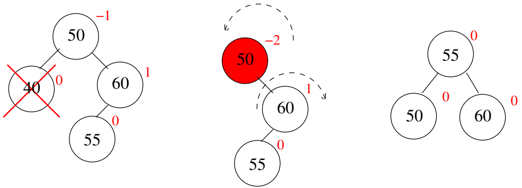
Remoção - Caso Dir-Dir - bal(filho)=0 - Novo Caso!!



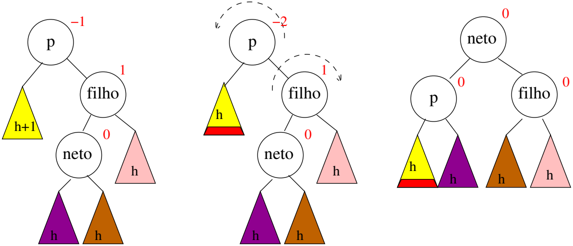
De forma generica:



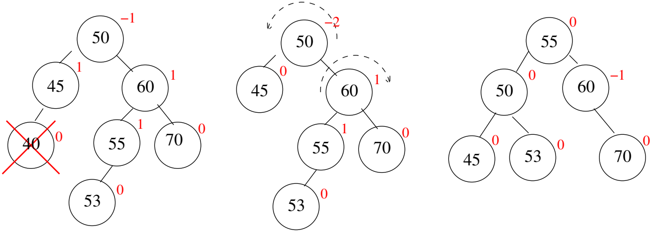
Remoção - Caso Dir-Esq - bal(neto)=0 - Novo Caso!!



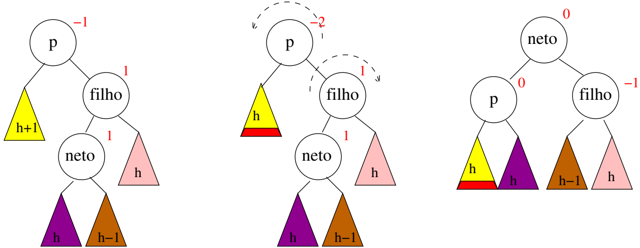
De forma generica:



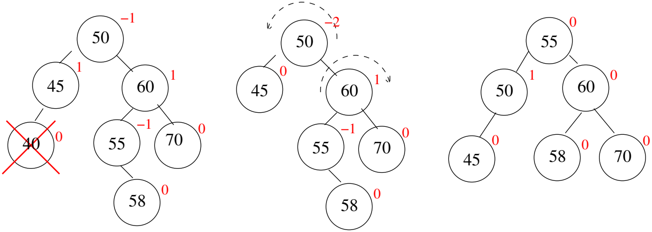
Remoção - Caso Dir-Esq - bal(neto)=1



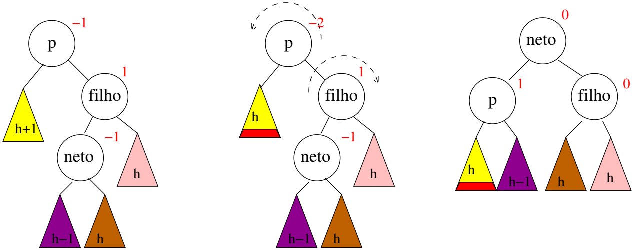
De forma generica:



Remoção - Caso Dir-Esq - bal(neto)=-1



De forma generica:



Balanceamento quando *bal*(*a*) é -2

```
f = direita(a)
se bal(f) == -1
    rotacaoEsquerda(a)
    bal(a) = bal(f) = 0
senão se bal(f) == 0
    rotacaoEsquerda(a)
    bal(a) = -1
    bal(f) = 1
```

```
senão /* bal(f) == 1) */
    neto = esquerda(f)
    rotacaoDireita(f)
    rotacaoEsquerda(a)
    se bal(neto) == 1
        bal(a)= 0
        bal(f) = -1;
    senão se bal(neto) == -1
        bal(a) = 1;
        bal(f) = 0;
    senão /* bal(neto) == 0 */
        bal(a) = bal(f) = 0
    bal(neto) = 0;
```

Remoção com desbalanceamento à esquerda

Simule os cinco casos de desbalanceamento quando a subárvore maior é o filho à esquerda do nodo desbalanceado.

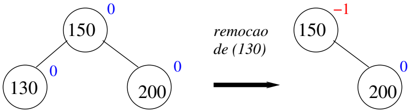
Balanceamento quando bal(a) é $+2$

```
f = esquerda(a)
se bal(f) == 1
    rotacaoDireita(a)
    bal(a) = bal(f) = 0
senão se bal(f) == 0
    rotacaoDireita(a)
    bal(a) = 1
    bal(f) = -1
```

```
senão /* se bal(f) == -1 */
    neto = direita(f)
    rotacaoEsquerda(f)
    rotacaoDireita(a)
    se bal(neto) == 1
        bal(a) = -1;
        bal(f) = 0;
    senão se bal(neto) == -1
        bal(a) = 0;
        bal(f) = 1;
    senão /* bal(neto) == 0 */
        bal(a) = bal(f) = 0
    bal(neto) = 0
```

Quando parar a atualização de balanceamentos

Quando a altura da árvore resultante é a mesma de **ANTES** da remoção.



Quando a árvore muda de altura?

- quando o balanceamento torna-se zero após a remoção
  - $bal(n) = -1 \rightarrow bal(n) = 0$
    a altura da subárvore esquerda era maior, mas diminui de 1
  - $bal(n) = 1 \rightarrow bal(n) = 0$
    a altura da subárvore direita era maior, mas diminui de 1
- Observe o que acontece nas alturas das subárvores nos casos de balanceamento

Implementação da inserção na árvore AVL

- `Remove( valorDaChave, *raiz )`
  altera a árvore com o nodo contendo o `valorDaChave` removido, caso ele exista
- chama uma função recursiva
  `ArvAVL removeR( ArvAVL nodoRem, ArvAVL nodoAtual, int *mudouH )`
  Parâmetro `mudouAltura`:
  `true`: se mudou a altura da subárvore com raiz no `nodoAtual` `false`: caso contrário

```c
void remove( ItemAvl k, ArvAVL *raiz ){
  ArvAVL nodoK, nodoRem;
  ItemAvl itemRem;
  int mudouH;

  /* busca nodo que contem chave k */
  nodoK = busca( k, *raiz );
  if( nodoK == nodoNull )
    return;

  /* busca nodo com dados que vao substituir chave k que sera' removida */
  if( nodoK->dir == nodoNull || nodoK->esq == nodoNull )
    nodoRem = nodoK;
  else
    nodoRem = buscaMaior( nodoK->esq );
  itemRem = nodoRem->item;

  /* remove nodoRem da arvore */
  /* nodoRem e' folha ou tem um unico filho */
  *raiz = removeR( nodoRem, *raiz, &mudouH );
  if( itemRem != k)
    nodoK->item = itemRem;
  return;
}
```

```
ArvAVL removeR( ArvAVL nodoRem, ArvAVL p, int *mudouH ){
  ArvAVL filho;

  if( p == nodoRem ){          /* remove nodoRem folha ou com 1 filho */
    if( p->dir != nodoNull ) filho = p->dir; else filho = p->esq;
    free( p );
    *mudouH = TRUE;
    return filho;
  }
  else if( nodoRem->item < p->item ){
    p->esq = removeR( nodoRem, p->esq, mudouH );
    if( *mudouH ){
      p->bal--;
      if( p->bal == -2 )       /* mesmo balanceando a altura da subarv. muda */
        balanceia( &p );
      if( p->bal != 0 )
        *mudouH = FALSE;       /* a remocao nao altera a altura da subarv. */
    }
  }
  else {
    p->dir = removeR( nodoRem, p->dir, mudouH );
    if( *mudouH ){
      p->bal++;
      if( p->bal == 2 )
        balanceia( &p );
      if( p->bal != 0 )
        *mudouH = FALSE;
    }
  }
  return p;
}
```